VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Aplikace pro snímání, předzpracování a uložení měřených údajů z LCD

Application for Capture, Preprocessing and Storage of Measured Data from LCD Screens

2016 Jakub Piško

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jakub Piško**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Aplikace pro snímání, předzpracování a uložení měřených údajů z LCD
Application for Capture, Preprocessing and Storage of Measured Data from LCD Screens

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je navrhnout a realizovat aplikaci, která bude pomocí integrovaného fotoaparátu mobilního zařízení snímat obraz hodnot na LCD displeji, umožnit uživateli definovat oblasti a údaje, které obsahují, převést obraz na výslednou hodnotu resp. hodnoty (a případně je předzpracovat) a umožnit hodnoty uložit resp. exportovat do externího URL.

1. Prostudujte techniky optického rozpoznávání znaků (OCR) a popište jak fungují.
2. Popište, jaké možnosti OCR a jaké knihovny pro OCR jsou k dispozici na platformě Android.
3. Analyzujte požadavky, které budou kladeny na aplikaci a navrhněte uživatelské rozhraní.
4. Implementujte vlastní knihovnu a aplikaci, která bude data snímat a předzpracovávat.
5. Otestujte funkčnost aplikace a vyhodnoťte dosažené výsledky.

Seznam doporučené odborné literatury:

[1] Steele, J., To, N.: The Android Developer's Cookbook: Building Applications with the Android SDK, Addison-Wesley Professional, 2010, ISBN-13: 978-0321741233
[2] Meier, R.: Professional Android 4 Application Development, Wrox, 2012, ISBN-13: 978-1118102275[1] Steele, J., To, N.: The Android Developer's Cookbook: Building Applications with the Android SDK, Addison-Wesley Professional, 2010, ISBN-13: 978-0321741233

[3] Hashimi, S.: Pro Android 2, Apress, 2010, ISBN-13: 978-1430226598
[4] John C. Russ: The Image Processing Handbook, Sixth Edition, CRC Press, 2011, ISBN 1439840636.
[5] Tesseract Open Source OCR Engine [online] [cit. 2015-09-05] Dostupné z: https://github.com/tesseract-ocr

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Moravec, Ph.D.**

Datum zadání: 01.09.2015

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 29. apríla 2016 .....

Súhlasím so zverejnením tejto bakalárskej práce podľa požiadavkou čl. 26, odst. 9 Študijného a skúšobného plánu pre štúdium v bakalárskych programoch VŠB-TU Ostrava.

V Ostrave 29. apríla 2016 .....

Veľmi rád by som sa poďakoval všetkým, ktorí mi pomáhali pri vzniku tejto práce. Hlavne vedúcemu mojej bakalárskej práce, Ing. Pavel Moravec, Ph.D. za cenné rady pri konzultáciách. Ďalej by som rád poďakoval svojej rodine za podporu počas štúdia.

Abstrakt

Podstatou tejto bakalárskej práce bolo navrhnúť a implementovať aplikáciu pre operačný systém Android, ktorej úlohou je snímať údaje z LCD displeja a následne ich predspracovať a uložiť. Práca obsahuje teoretickú časť popisujúcu princíp fungovania optického rozpoznávania znakov, jeho komponenty a časté problémy. V práci sú preskúmané a porovnané existujúce OCR knižnice. Praktická časť zahŕňa implementáciu, testovanie aplikácie pomocou dostupných nástrojov a ladenie prvkov, ktoré ovplyvňujú úspešnosť rozpoznávania.

Kľúčové slová: aplikácia, optické rozpoznávanie znakov, predspracovanie, LCD displej, Android

Abstract

The goal of this bachelor thesis is to design and implement Android application, whose task is to capture data from a LCD screen, preprocess and save them. The thesis contains theoretical part describing the principle of operation of optical character recognition, its components and common problems. It explores and compares existing OCR libraries. Practical part includes implementation, testing of the application using available tools and fine-tuning of parameters, which influence the recognition success.

Key Words: application, optical character recognition, preprocessing, LCD screen, Android

Obsah

Zoznam použitých skratiek a symbolov

AF	–	Application Framework
API	–	Application Programming Interface
ARM	–	Advanced RISC Machine
GSM	–	Global System for Mobile Communications
JPEG	–	Joint Photographic Experts Group
JSON	–	JavaScript Object Notation
JVM	–	Java Virtual Machine
LCD	–	Liquid-Crystal Display
NDK	–	Native Development Kit
OCR	–	Optical Character Recognition
OS	–	Operating System
PSM	–	Page Segmentation Mode
RAM	–	Random Access Memory
RISC	–	Reduced Instruction Set Computer
TIFF	–	Tagged Image File Format
UI	–	User Interface
URL	–	Uniform Resource Locator
USD	–	United States Dollar
VM	–	Virtual Machine
XML	–	Extensible Markup Language

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov zdrojového kódu

Úvod

Táto bakalárska práca je zameraná na technológiu optického rozpoznávania znakov, ktorá sa za posledné polstoročie výrazne vyvinula a v dnešnej dobe má široké využitie. Či sa jedná o digitalizáciu starých dokumentov alebo čítanie a prekladanie textu v reálnom čase, potrebujeme spracovať analogový text do digitálnej podoby.

Mnohé zariadenia podporujú šírenie zobrazovaných údajov pomocou WiFi alebo bluetooth, tie sú však často limitované lokálnou sieťou a exportom dát pomocou špecializovaného softvéru. Vytvorená aplikácia tieto limity odstráni a umožní jej užívateľovi šíriť prečítané dáta cez internet. Údaje budú zaslané na preddefinovanú URL vo formáte JSON, pričom na ukladanie môžeme využiť dedikovaný server alebo voľne dostupné služby. Takto uložené dáta môžeme ďalej spracovať napríklad za účelom generovania štatištík alebo jednoducho archivovať.

Aplikácia má veľký potenciál využiteľnosti, pretože má minimálne požiadavky na hardvér a softvér. Mobilný telefón má dnes takmer každý a spôsob, akým aplikáciu využije, je limitovaný jeho predstavivosťou. V prípade nedostatočnej funkcionality je možné aplikáciu jednoduchým spôsobom rozšíriť o ďalšie znaky a znakové sady.

Práca je rozdelená na 3 hlavné časti. V prvej popíšeme princíp fungovania OCR, jeho vývoj za posledné roky, komponenty z ktorých bežný OCR systém pozostáva a časté chyby pri rozpoznávaní. Ďalšia časť obsahuje popis a porovnanie aktuálne dostupných softvérových knižníc, ktoré môžeme pri rozpoznávaní využiť. Posledná časť sa zaoberá návrhom a implementáciou Android aplikácie, ktorá analógové dáta prečíta a spracuje. Táto kapitola taktiež popisuje ladenie aplikácie, ktoré je nevyhnutné pre dosiahnutie vysokej úspešnosti rozpoznávania.

1 Optické rozpoznávanie znakov – OCR

Optické rozpoznávanie znakov je technológia na prevod digitálnych textových obrázkov do textovej podoby. Umožňuje nám strojové rozpoznávanie textu, ktorý môžeme následne ďalej spracovať. V tejto kapitole sme čerpali z [4].

1.1 História OCR

Už na žačiatku 19. storočia začali prvé pokusy vytvoriť zariadenia, ktoré by umožnili slepým ľuďom čítať. Moderná verzia OCR prišla až v 40. rokoch 20. storočia, s nástupom digitálneho počítača. Odvtedy bol veľký záujem o ďalší vývoj, keďže možnosti využitia sú veľmi obrovské.

1.1.1 Začiatky OCR

Keď počas 50. rokov 20. storočia začala technologická revolúcia naberať na obrátkach, elektronické spracovanie dát bolo čoraz dôležitejšie. Vstupné dáta vtedajších počítačov sa zadávali pomocou diernych štítkov a bolo potrebné vyvinúť spôsob, ktorý by znížil náklady na spracovanie zvyšujúceho sa počtu dát. V tom istom čase sa technológia strojového čítania vyvinula dostatočne, aby bola použiteľná v reálnych aplikáciách. Neskôr, v polovici 50. rokov, sa na trhu objavili prvé, komerčne dostupné OCR systémy.

Prvý OCR systém bol inštalovaný magazínom Reader's Digest, kde sa používal na konverziu písaných správ predaja do diernych štítkov, ktoré následne spracoval počítač.

1.1.2 Prvá generácia OCR

Komerčné OCR systémy, ktoré sa objavili medzi rokmi 1960 a 1965, môžeme nazvať prvou generáciou OCR. Táto generácia OCR systémov je charakteristická obmedzeným tvarom písmen, ktoré dokázala čítať. Tieto znaky boli navrhnuté špecificky pre strojové čítanie, a mnohé z nich nevyzerali prirodzene. Po čase sa objavili systémy, ktoré dokázali pracovať s väčším počtom znakových sád. Počet písiem, ktoré boli čitateľné, bol limitovaný použitým spôsobom rozpoznávania a priraďovania šablón. Šablóny následne porovnávali znaky z obrázku a prototypy znakov pre každú znakovú sadu.

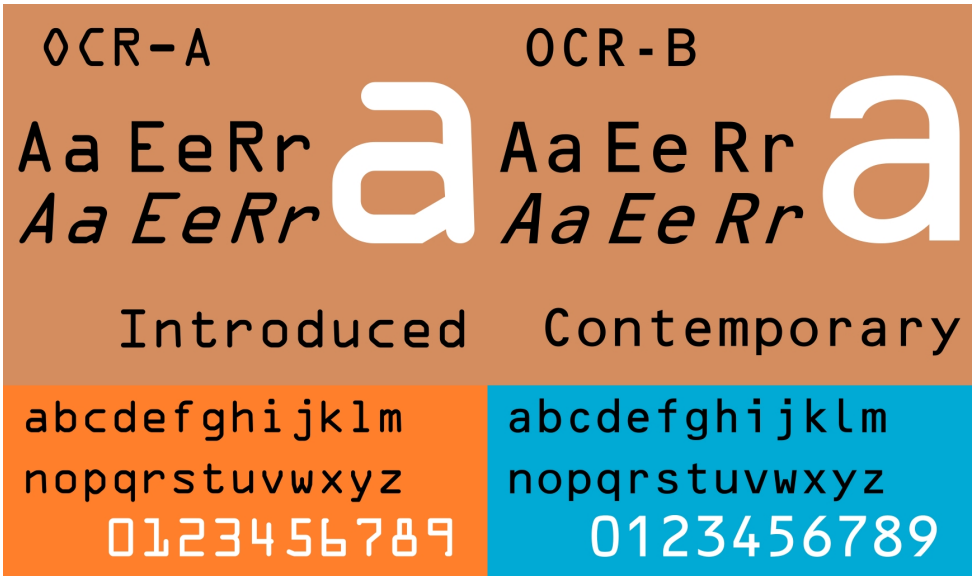
1.1.3 Druhá generácia OCR

Rozpoznávacie systémy druhej generácie sa objavili koncom 60. a začiatkom 70. rokov. Tieto systémy boli schopné rozpoznať bežné strojom tlačené znaky a taktiež mali do určitej miery podporu rozpoznávať text písaný rukou. Znaková sada rukou písaného textu bola limitovaná na čísla, niektoré písmená a znaky.

Prvý známy systém tohto druhu bol IBM 1287, ktorý bol predstavený na svetovej výstave v roku 1965 v New Yorku. V tom istom čase vyvinula Toshiba automatický systém zoraďovania

poštových čísiel a spoločnosť Hitachi vytvorila prvý OCR systém, ktorý bol dostatočne výkonny a cenovo dostupný pre verejnosť.

V tomto období sa tiež výrazne pracovalo na štandardizácii. V roku 1966 bola vypracovaná štúdia zameraná na potreby OCR a jej výsledkom bol americký štandard – OCR-A [1]. Tento štandard definoval písmo pre rozpoznávanie, ktoré bolo primárne určené pre strojové čítanie, ale muselo tiež byť ľahko čitateľné pre ľudí. Európsky štandard, nazvaný OCR-B [2], bol vytvorený v roku 1968. Európska verzia obsahovala písmo, ktoré vyzerá prirodzenejšie ako americká verzia. Nasledovalo niekoľko pokusov spojiť oba štandardy do jedného, používaného na celom svete, tieto pokusy však vo veľkej miere zlyhali a výsledkom boli systémy, ktoré podporovali obe verzie štandardu.



Obr. 1: Porovnanie OCR-A a OCR-B, prevzaté z [3]

1.1.4 Tretia generácia OCR

Tretia generácia OCR systémov, ktorá sa objavila v polovici 70. rokov, mala za úlohu riešit problémy při čítaní dokumentov nízkej kvality a rozpoznávaní rukou písaného textu. Nízka cena a vysoký výkon boli určujúce faktory, ktoré riadili vývoj a ovplyvnili aj hardvérové časti OCR systémov.

Aj napriek sofistikovaným systémom OCR, ktoré sa začali objavovať na trhu, bol veľký dopyt po jednoduchých zariadeniach, ktoré boli určené na špecifické úlohy. V čase, kedy boli osobné počítače stále drahá záležitosť a softvérová výbava na úpravu textu limitovaná, boli jednoducho navrhnuté OCR zariadenia veľmi užitočné. Často sa používali pri úprave textu, kedy bol hrubý návrh napísaný na písacom stroji, ktorého výstup mal rovnomerné rozmiestnenie textu a bol ľahko strojovo čitateľný. Takýmto spôsobom bolo možné výrazne znížiť náklady na potrebný softvér a hardvér.

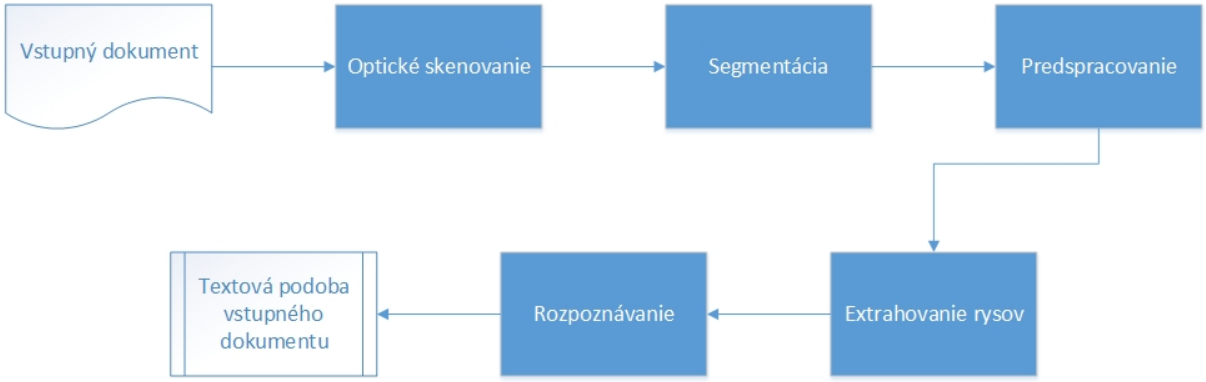
1.1.5 Štvrtá generácia OCR

Komerčné OCR systémy boli dostupné už v 50. rokoch, napriek tomu sa ich do roku 1986 predalo iba niekoľko tisíc kusov. Hlavným dôvodom bola stále vysoká cena vybavenia. To sa však zmenilo, keď začali byť dostupné OCR systémy ako softvérové balíčky a ich predaj výrazne stúpol. V dnešnej dobe sa každý deň predá niekoľko tisíc hardvérových OCR systémov a ich cena sa neustále znižuje. Existuje aj množstvo softérových OCR systémov, ktoré sú výrazne lacnejšie ako ich hardvérová obdoba. Niektoré z nich sú dokonca voľne dostupné – pár ich popíšeme v nasledujúcej kapitole.

1.2 Ako OCR funguje

Hlavný princíp automatického rozpoznávania znakov je založený na rozpoznávaní znakových šablón, ktoré musíme systém naučiť. Tento proces pozostáva z definovania množiny čitateľných znakov a ich pravdepodobného umiestnenia v texte. Následne učenie prebieha formou ukázania sady znakov systému, kedy si systém vytvorí internú reprezentáciu znakových prototypov s popisom každého znaku. Neskôr sa pri rozpoznávaní porovnávajú čítané znaky s ich prototypmi a vyberie sa ten, ktorý najviac zodpovedá prototypu. Znaky, ktoré systém nepozná, sa priradia k ich najlepšie pasujúcemu prototypu.

Vo väčšine komerčných OCR systémov prebieha učebný proces ešte počas výroby. Niektoré systémy obsahujú nástroje umožňujúce trénovanie ďalších znakových sád. [5]



Obr. 2: Komponenty OCR systému

1.3 Komponenty OCR

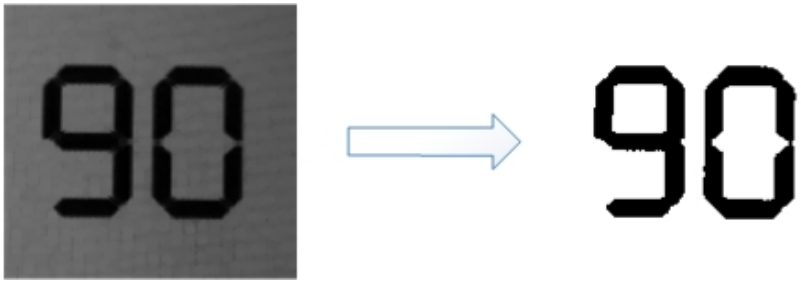
Typický OCR systém pozostáva z niekoľkých kompenentov. Obrázok 2 znázorňuje bežnú konfiguráciu komponentov. Prvým krokom procesu rozpoznávania je digitalizácia analógového dokumentu pomocou optického skeneru. Nasleduje vyhľadanie textových blokov, z ktorých sa extrahujú jednotlivé znaky. Extrahované znaky môžu podstúpiť predspracovanie, ktoré eliminuje vizuálny šum a odstráni vizuálne artefakty. Predspracované znaky môžu byť použité pri extrahovaní rysov.

Identifikácia znakov prebieha porovnávaním rysov znaku s definovanými prototypmi, následne sa použijú kontextuálne informácie k rekonštrukcii slov a čísiel z pôvodného textu.

1.3.1 Optické skenovanie

Tento proces prevádza analógový dokument do formy digitálneho obrázku. Väčšina OCR systémov pracuje s čierno-bielymi obrázkami, preto je ideálny optický senzor, ktorý zaznamenáva intenzitu svetla ako odtieň šedej. V prípade senzora zachytávajúceho farbu je potrebné obrázok previesť do binárnej formy. Táto konverzia môže prebiehať aj priamo v senzore. Ušetrí sa tak pamať a výpočetný výkon – nazýva sa prahovanie.

Proces prahovania znázornený v obrázku 3 je dôležitá súčasť skenovania, ktorá priamo ovplyvňuje kvalitu výsledného rozpoznávania. Tento proces má za úlohu oddeliť text od pozadia. Často sa používa jednoduché prahovanie, v ktorom sa určí pevný prah a následne sa obrázok spracuje spôsobom, kedy sa jednotlivé pixely obrázku prevádzajú na čiernu a bielu farbu podľa toho, či sa ich aktuálna farba nachádza nad alebo pod definovaným prahom. Takéto jednoduché prahovanie je možné použiť v prípade, že vstupný obrázok má dostatočne vysoký kontrast farieb medzi textom a pozadím. V mnohých prípadoch ale takéto jednoduché prahovanie nestačí, keď je množina kontrastov v dokumente príliš veľká. V týchto prípadoch je potrebné použiť sofistikovanejšie metódy prahovania, ktoré zaistia lepšie výsledky.



Obr. 3: Prahovanie vstupného obrázka

1.3.2 Segmentácia

Segmentácia je proces, ktorý v digitálnom dokumente vyhľadá oblasti s textom. Väčšina OCR algoritmov rozpoznáva text ako jednotlivé znaky, ktoré sú definované ako navzájom prepojené čierne oblasti. Takýto algoritmus je postačujúci pre základné znaky. Problém ale nastáva, ak sa znaky navzájom dotýkajú alebo sú tvorené z viacerých oddelených častí. Hlavné problémy segmentácie môžu byť rozdelené do 2 skupín:

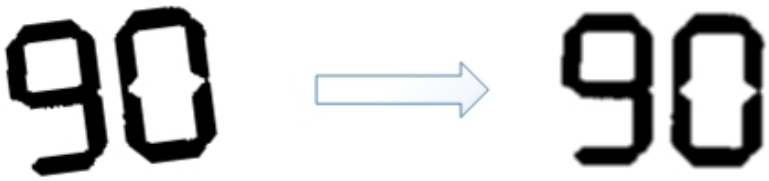
- extrahovanie dotýkajúcich sa a fragmentovaných znakov: viacero spojených znakov môže byť rozpoznaných ako jeden, naopak časti znaku môžu byť interpretované ako samostatné znaky
- rozlíšenie šumu: diakritika môže byť rozpoznaná ako šum a naopak

1.3.3 Predspracovanie

Výsledný dokument po skenovaní môže obsahovať určité množstvo šumu, ktoré je potrebné odstrániť. Po zoskenovaní a následnom prahovaní môžu ostať niektoré znaky rozmazané alebo rozdelené. Tieto nedostatky musíme odstrániť pomocou predspracovania, pretože by ovplyvnili úspešnosť rozpoznávania. Predspracovanie najčastejšie zahŕňa vyhladenie a normalizáciu.

Vyhladenie prebieha formou vyplnenia znaku, ktoré odstráni malé medzery a diery. Následne sa pomocou zúženia zníží šírka čiary, ktorá tvorí znak. Najbežnejšie sa vyhladenie binárneho dokumentu vykonáva aplikovaním týchto techník po blokoch, kým nie je vyhladený celý dokument.

Normalizácia znázornená v obrázku 4 je aplikovaná za účelom získania znakov, ktoré majú rovnakú veľkosť a rotáciu. Na opravu rotácie je najskôr potrebné zistiť uhol odchýlky, o ktorý potrebujeme obrázok otočiť. Tento uhol je možné zistiť pomocou Houghovej transformácie zo strán alebo riadkov textu, nie však z jednotlivých znakov. Rotáciu samostatného znaku môžeme zistiť iba z rozpoznaného znaku.



Obr. 4: Normalizácia vstupného obrázka

1.3.4 Extrahovanie rysov

Úlohou tohoto procesu je zachytiť základnú charakteristiku znakov, ktorá popisuje abstraktné rysy znaku ako čiary, ich prekrývanie, prázdne bloky a podobne. Techniky extrahovania môžeme rozdeliť do 3 skupín:

- distribúcia bodov
- transformácia a expanzia
- analýza štruktúr

Optimálny spôsob extrahovania skupiny rysov môžeme určiť podľa množstva šumu v dokumente, jeho deformácie, náročnosti na implementáciu a používanie. Pri výbere musíme zohľadniť nasledujúce kritériá:

- robustnosť
  - šum

  - deformácia
  - rotácia
  - variácia štýlu znakov
- jednoduchosť používania
  - rýchlosť rozpoznávania
  - náročnosť implementácie
  - nezávislosť na externom predspracovaní

1.3.5 Klasifikácia

Klasifikácia je proces identifikácie prečítaných znakov. Pri klasifikácii sa používajú znaky reprezentované číslami vo vektore rysov.

Párovanie

Párovanie je skupina techník založených na podobnosti výsledkov meraní vzdialeností medzi vektorom rysov popisujúcim znak a prototypom znaku. Najčastejšie sa na výpočet vzdialenosti používa Euklidova metrika. Klasifikácia pomocou minimálnej vzdialenosti funguje dobre, ak sú prototypy znakov navzájom dostatočne odlišné.

Ak sa ako vstup klasifikácie použije celý znak bez extrahovaných rysov, uplatníme postup korelácie, ktorý páruje znaky na základe vypočítanej vzdialenosti medzi obrázkom znaku a prototypom.

Optimálna štatistická klasifikácia

Tento proces rozpoznáva znaky na základe pravdepodobnosti. Používa klasifikačnú schému, ktorá je optimálna, pretože jej priemerná chybovosť je nízka.

Najčastejšie sa používa Bayesova klasifikácia, ktorá minimalizuje výslednú priemernú stratu. Pravdepodobnosť úspešného rozpoznávania neklasifikovaného znaku voči prototypu p sa počíta pre všetky prototypy $p = 1..N$. Následne sa vyberie prototyp, ktorý má najvyššiu pravdepodobnosť byť správny.

Aby bol tento proces optimálny, potrebujeme vedieť pravdepodobnosť výskytu každého prototypu. V prípade, že je pravdepodobnosť výskytu neznáma, môžeme predpokladať, že sa jednotlivé prototypy vyskytujú v texte rovnomerne. Úspešnosť Bayesovej klasifikácie stúpa priamo úmerne s presnosťou predpokladaného výskytu.

Neurónové siete

Rozpoznávanie znakov je možné aj pomocou neurónových sietí, konkrétne použitím siete so spätným šírením. Tá sa skladá z niekoľkých vrstiev navzájom prepojených neurónov. Ako vstup siete

použijeme vektor rysov, ktorý sa spracuje jednotlivými neurónmi pomocou nelineárnej funkcie. Počas trénovania siete sa váhy každého neurónu menia, pokiaľ nedosiahneme predpokladaný výstup. Problémom neurónových sietí v OCR je ich limitovaná predvídateľnosť a všeobecnosť. Ich prednosťou je zase adaptívny charakter.

1.3.6 Pospracovanie — zoskupovanie

Výsledkom samotného rozpoznávania znakov dokumentu je množina individuálnych znakov, z ktorých je potrebné vytvoriť slová a čísla. Tento proces spájania znakov sa nazýva zoskupovanie. Zoskupovanie znakov do slov prebieha na základe ich umiestnenia v dokumente. Znaky, ktoré sú dostatočne blízko pri sebe, sú zoskupené.

Zoskupovanie znakovej sady, ktorá má rovnomerné rozloženie znakov, je celkom jednoduché. Medzery medzi slovami sú väčšinou väčšie ako medzery medzi znakmi, preto je zoskupovanie vysoko úspešné. Problémy nastávajú pri rukou písanom písme alebo keď je text nakrivo.

1.4 Detekovanie chýb a ich oprava

Až po zoskupenie sa sa každý znak spracovával samostatne, a kontext, v ktorom sa znak nachádzal, sa nebral do úvahy. Avšak ani pokročilé OCR systémy nevedia rozpoznať všetky znaky s absolútnou úspešnosťou. Musíme počítať s určitou mierou chybovosti. Niektoré z týchto chýb sú detekovateľné a dokonca opraviteľné, ak použijeme kontext, v ktorom sa znak nachádza.

Jeden zo spôsobov, ako chyby detekovať, je vytvorenie množiny znakov, ktoré sa môžu vyskytnúť v texte spolu. Túto množinu môžeme definovať napríklad ako syntax slova. Efektívnou metódou detekovania a opravy chýb sú slovníky. Rozpoznané slovo sa porovná so slovami slovníka a ak sa hľadané slovo nenašlo, predpokladáme chybu. Toto chybné slovo môžeme zameniť za slovo zo slovníka, ktoré je mu najpodobnejšie. Vektory rysov používané pri klasifikácii je možné použiť na detekovanie nesprávnych znakov v slove. Ak sa hľadané slovo v slovníku nachádza, je pravdepodobné, že slovo bolo rozpoznané správne. Výskyt v slovníku ale negarantuje úspešnosť rozpoznávania, pretože mohlo dôjsť k chybám, ktoré vytvorili z neplatného slova platné – takéto chyby nie je možné detekovať ani opraviť pomocou slovníka. Nevýhoda vyhľadávania v slovníku je jeho časová náročnosť, ktorá stúpa s jeho komplexnosťou.

2 OCR knižnice

2.1 Tesseract

Tesseract [6] je open source OCR engine, dostupný pod licenciou Apache v2.0. Je napísaný v C++ a kompilovateľný pre veľké množstvo operačných systémov. Pôvodne bol vyvíjaný od roku 1985 ako proprietárny softvér spoločnosťou Hewlett Packard. V roku 2005 bol vydaný ako open source a ujala sa ho spoločnost Google, ktorá projekt sponzoruje od roku 2006. Tesseract je možné používať priamo, ako nezávislú aplikáciu na rozpoznávanie textu, alebo ako súčasť aplikácie vo forme API. Ako nezávislá aplikácia nemá oficiálne užívateľské rozhranie. To je možné stiahnuť od tretích strán.

Tesseract vo verzii 2 vedel spracovať iba vstupné obrázky vo formáte TIFF, ktoré obsahovali jeden riadok textu. Táto verzia nepodporovala analýzu rozloženia textu, preto viacriadkový text produkoval nečitateľný výstup. Vo verzii 3 pribudla podpora detekcie rozloženia textu a hOCR, ktorá reprezentuje výstupný formátovaný text používaný v OCR. Ten obsahuje informácie ako kódovanie, štýl a rozloženie textu. Pribalená knižnica Leptonica zaisťuje podporu ďalších formátov obrázkov.

Počiatočné verzie vedeli spracovať iba text v anglickom jazyku. Verzia 2 pridala podporu západoeurópskych jazykov, verzia 3 výrazne rozšírila počet podporovaných jazykov a pribudla možnost trénovania vlastného jazyka, resp. znakovej sady.

2.2 OCRopus

OCRopus [7] je kolekcia nástrojov napísaných v jazykoch C++ a Python, určených na analýzu dokumentov. Používa modulárny dizajn založený na pluginoch. Tie umožňujú ľahko zamienať jednotlivé kompomenty používané pri rozpoznávaní textu. OCRopus je primárne vyvíjaný pre Linux, dá sa však používať aj na Mac OS X. OCRopus je licencovaný pod Apache v2.0, ale chýba mu podpora operačného systému Android, preto je pre prácu nepoužiteľný.

2.3 Asprise OCR

Spoločnosť Asprise ponúka veľké množstvo nástrojov na spracovanie obrázkov. Ich OCR SDK [8] je dostupné pre viacero programovacích jazykov, hlavne C++, C#, Python a Java. Hlavnou prednosťou je vysoký výkon API knižnice, ktorú je možné využívať na takmer akejkoľvek platforme. Jej obsiahla funkcionalita zahŕňa extrahovanie textu a barcode informácií zo zoskenovaných dokumentov a obrázkov. Zvláda tiež konverziu obrázkov množstva rôznych formátov do čitateľných dokumentov. Riešenia spoločnosti Asprise majú oficiálnu podporu a dá sa teda očakávať, že používanie Asprise OCR je vhodné aj pre veľké projekty. Pre túto prácu je však knižnica nevhodná. Hlavne kvôli licenčným poplatkom, ktoré sa v čase písania začínajú na úrovni 5000 USD a viac.

2.4 TessTwo

TessTwo [9] je open source Java API, ktoré obaľuje knižnicu Tesseract a umožňuje jej používanie pod operačným Android. Podporuje aktuálne najnovšiu verziu knižníc Tesseract a Leptonica. Vyžaduje Android v2.3 a vyšší, čo mierne limituje množstvo zariadení, na ktorých je táto knižnica použiteľná.

3 Android

Android nie je len názov operačného systému, ktorý aktuálne dominuje na trhu mobilných zariadení. Je to platforma pozostávajúca z množstva dobre navrhnutých vrstiev, ktoré prepájajú kernel Linuxu a aplikácie bežiace na každom zariadení. Android je dostupný pre architektúry ARM a x86, teoreticky je ale možné ho preportovať na akúkoľvek hardwarovú platformu podporovanú Linuxom.

3.1 Platforma Android

Obr. 5: Platforma Android

Operačný systém Android je postavený nad Linuxovým kernelom. Počiatočná verzia Android 1.0 používala kernel verzie 2.6.27, ktorý je do veľkej miery upravený a spravovaný spolkom Open Handset Alliance. Upravený kernel ponúka vývojárom abstrakciu hardvéru a ovládače pre časti smartfónov ako displej, fotoaparát, GSM modem, ako aj správu napájania a nízkoúrovňovú bezpečnosť. K častiam operačného systému, ako napríklad správa napájania a nízkoúrovňová bezpečnosť, majú prístup iba výrobcovia mobilných zariadení, nie vývojári aplikácií.

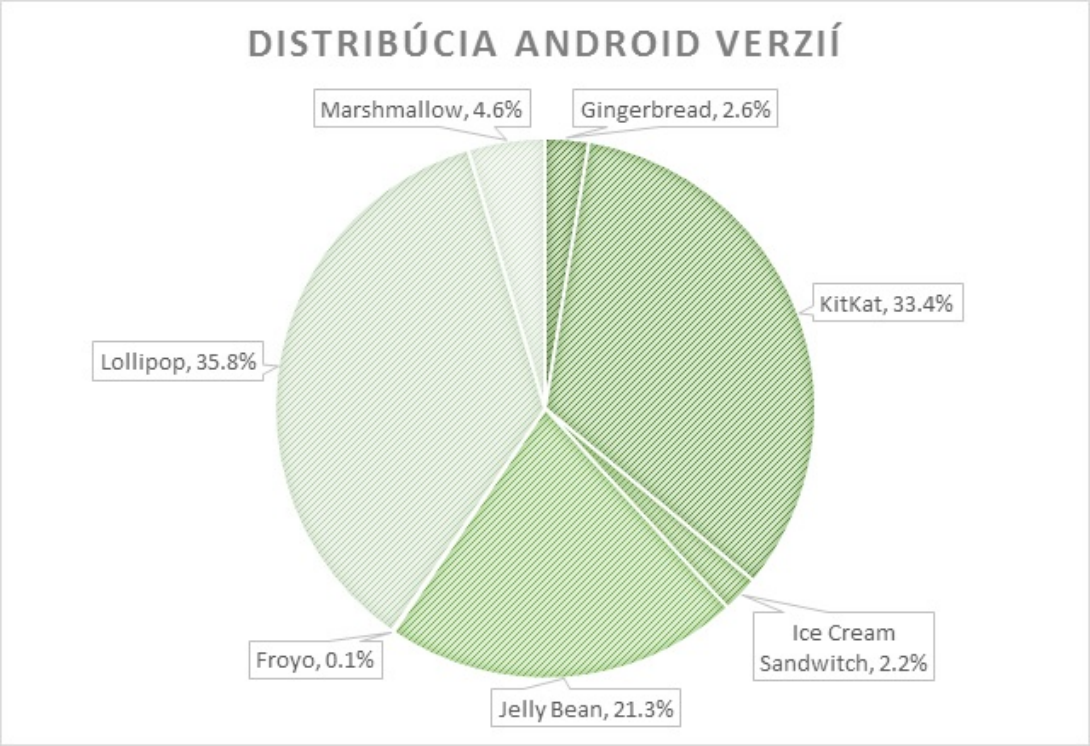
Android poskytuje množstvo knižníc, ktoré sú využívané rôznymi komponentami platformy. Štandardné knižnice sú SQLite pre databázy, FreeType na renderovanie fontov, WebKit poháňajúci webové aplikácie a niekoľko mediálnych knižníc zabezpečujúcich prehrávanie audia a videa. Tieto knižnice sú poskytnuté ako abstraktná vrstva pomocou Android SDK, prípadne ich natívna verzia cez Android NDK.

Android runtime (behové prostredie) pozostáva z dvoch častí. Prvou je virtuálny stroj, v ktorom bežia aplikácie napásané pomocou SDK a druhou časťou je samotné Android SDK, ktoré je vývojárom poskytnuté ako skupina Java tried a frameworkov. Virtuálny stroj Dalvik je podobný ako JVM s tým rozdielom, že spúšta Dalvik bytecode namiesto Java bytecode. Dalvik bytecode sa generuje pri kompilácii a je optimalizovaný na nízku záťaž RAM pamäte. Výsledkom je Dalvik Executable format (.dex), ktorý sa dá spustiť cez Dalvik VM.

Application Framework je názov vysokoúrovňových API, ktoré vývojári používajú pri vytváraní aplikácií. Základom frameworku je Activity Manager, ktorý kontroluje životný cyklus aplikácií a poskytuje jednoduchú navigáciu medzi existujúcimi aktivitami. AF ponúka vývojárom:

- širokú škálu vizuálnych komponentov ako textové polia, tlačidlá a prepínače
- rozhranie Content Provider na komunikáciu a zdieľanie dát medzi aplikáciami

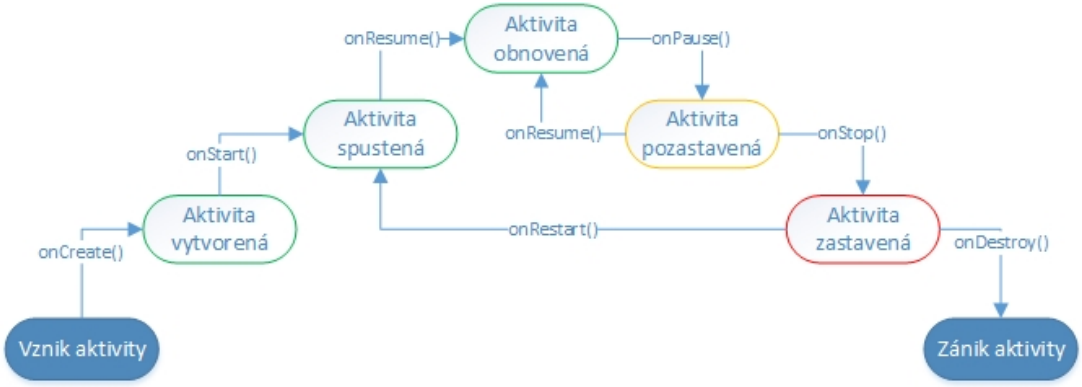
Ak má aplikácia dostatočné oprávnenia, môže napríklad čítať informácie o kontaktoch v zariadení. AF taktiež obsahuje Resource Manager, ktorý poskytuje prístup k zdrojom ako sú obrázky, súbory s rozložením užívateľského rozhrania a lokalizované znakové reťazce. [10]



Obr. 6: Distribúcia Android verzií, Apríl 2016, [11]

3.2 Životný cyklus aplikácie

Vývoj aplikácií pre mobilné platformy ako Android vyžaduje mierne odlišný spôsob programovania, pretože mobilná aplikácia je viac závislá na operačnom systéme v porovnaní s osobnými počítačmi. Pri vývoji musíme dbať na obmedzenia ako limitovaný výkon a úložný priestor. Okrem technických obmedzení je nutné brať do úvahy výrazne odlišný životný cyklus aplikácie, ktorý môže byť ľahko prerušený napríklad prichádzajúcim hovorom alebo vypnutím obrazovky.



Obr. 7: Životný cyklus Android aktivity

3.3 Projektová štruktúra

Projekty pre Android sa výrazne líšia od bežných Java aplikácií. Majú preddefinovanú štruktúru, kompilovateľný kód je oddelený od užívateľského rozhrania a taktiež tu nájdeme niekoľko automaticky generovaných súborov. Štruktúru čistého projektu popisuje tabuľka.

Tabuľka 1: Štruktúra Android projektu

Priečinky a súbory	Popis
\src	zdrojové kódy (aktivity a java triedy) aplikácie
\gen	automaticky generované triedy (trieda R s indexmi zdrojov)
\res	zdroje aplikácie
\drawable	obrázky pre rôzne veľkosti a rozlíšenia obrazovky
\layout	XML súbory rozloženia užívateľského rozhrania
\values	ostatné XML súbory s definíciami rôznuch elementov
\assets	ďalšie zdroje aplikácie
android-manifest.xml	metadáta, charakteristika a komponenty aplikácie

Okrem štandarných súborov a tried budeme musieť importovať knižnice openCV a tessTwo, ktoré sú vo forme zdrojových kódov a budeme ich kompilovať spolu s aplikáciou. Pred samotnou kompilácou potrebujeme nastaviť správnu verziu Android SDK. [12]

3.4 OpenCV pre Android

OpenCV je open-source knižnica poskytujúca niekoľko stoviek algoritmov zameraných na spracovanie obrazu a počítačové videnie. Je napísaná v C++ a poskytuje rozhrania pre množstvo programovacích jazykov, napríklad Python, Java a MATLAB. Knižnicu môžeme využívať na väčšine desktopových a mobilných OS. Pre vývoj na Android využijeme OpenCV4Android SDK.

Knižnica OpenCV sa líši funkcionalitou v závislosti od používanej verzie. Na načítanie a inicializáciu sa v Android SDK využíva externá aplikácia OpenCV Manager. Táto aplikácia vyberie vhodnú verziu OpenCV pre konkrétny hardvér zariadenia.

4 Analýza a návrh

V tejto časti popíšeme návrh a implementáciu aplikácie bežiacej pod operačným systémom Android.

4.1 Android 2.3

Pre potreby aplikácie bude minimálna verzia operačného systému Android určená použitými knižnicami a komponentami, pretože si v implementácii vystačíme so základnou API výbavou a nepotrebujeme pokročilú funkcionalitu novších verzií. Taktiež tým rozšírime množstvo zariadení, ktoré bude schopné aplikáciu využívať.

Knižnica OpenCV používaná na predspracovanie obrázkov určených na OCR nemá explicitne definovanú minimálnu verziu SDK a nie je teda závislá na verzii operačného systému. Je však možné nastaviť používanú verziu SDK a tým meniť poskytovanú funkcionalitu.

Knižnica TessTwo ktorá obaľuje funkcionalitu Tesseract OCR má pevne danú minimálnu verziu Android 2.3 a vyššie. Táto knižnica má určujúci faktor pri výbere verzie OS.

4.2 Fotoaparát

Fotoaparát obsahuje väčšina Android zariadení, je to však problematická časť systému a líši sa funkcionalitou závislou na konkrétnom modeli. Problémová je hlavne detekcia rotácie, ktorá na niektorých zariadeniach funguje správne, na niektorých čiastočne a na niektorých (ako napríklad zariadenie, na ktorom bola aplikácia testovaná) nefunguje vôbec. Je preto potrebné naimplementovať vlastné triedy, ktoré zaistia správnu funkčnosť rotácie obrázka. Tá je dôležitá pri správnom rozpoznávaní textu.

4.3 Úložný priestor

Nároky na úložný priestor sú minimálne, ale nie zanedbateľné. Pre správne spustenie aplikácie je potrebné vytvoriť kópiu naučeného fontu písma používaného knižnicou Tesseract do pamäte zariadenia. Jeho veľkosť sa pohybuje v rozpätí stoviek kilobajtov až jednotiek megabajtov. Ďalej musíme započítať veľkosť šablóny, ktorá obsahuje obrázkovú predlohu používanú na zarovnanie fotoaparátu a čítaného dokumentu, zoznam regiónov a informácie o každom regióne. Výstupné dáta môžeme zaslať na URL alebo uložiť do zariadenia. Sú vo formáte JSON a pozostávajú z popisu regiónu, z ktorého sa dáta čítali a hodnoty, ktorá bola prečítaná. Tieto výstupné súbory majú veľkosť rádovo jednotiek až desiatok kilobajtov.

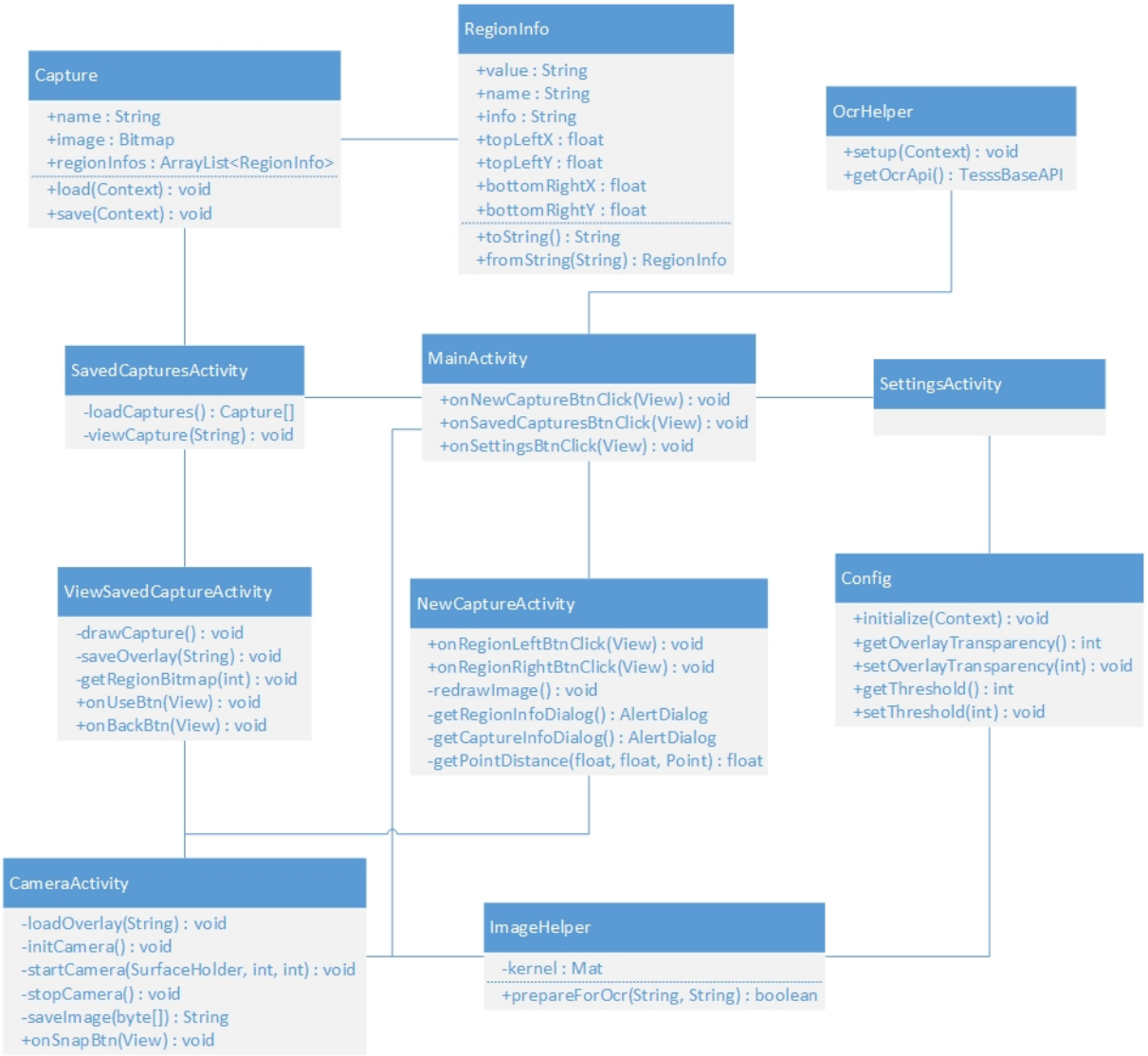
4.4 Prístup na Internet

Aplikácia ponúka možnosť zaslať prečítané údaje na webovú adresu, ktorá ich ďalej spracuje. Je preto potrebné internetové pripojenie, ktoré nemusí byť neustále aktívne, stačí, aby bolo dostupné po prečítaní údajov a po ich zaslaní sa používať nebude.

4.5 Návrh

Existuje niekoľko spoľahlivých a časom overených spôsobov, ako vyvíjať aplikácie. Každý má svoje prednosti a zápory. Niektorý je vhodné použiť v tíme vývojárov, iný je využiteľný jednotlivcom. Pre naše potreby a z nutnosti vytvoriť fungujúcu aplikáciu v relatívne krátkom čase je vhodný rapídny vývoj aplikácií. Tento spôsob vývoja je určený na tvorenie vysoko kvalitného a funkčného kódu za čo najkratší čas. Dovoľuje nám rozdeliť projekt na viacero častí, ktoré môžeme vyvíjať nezávisle na sebe.

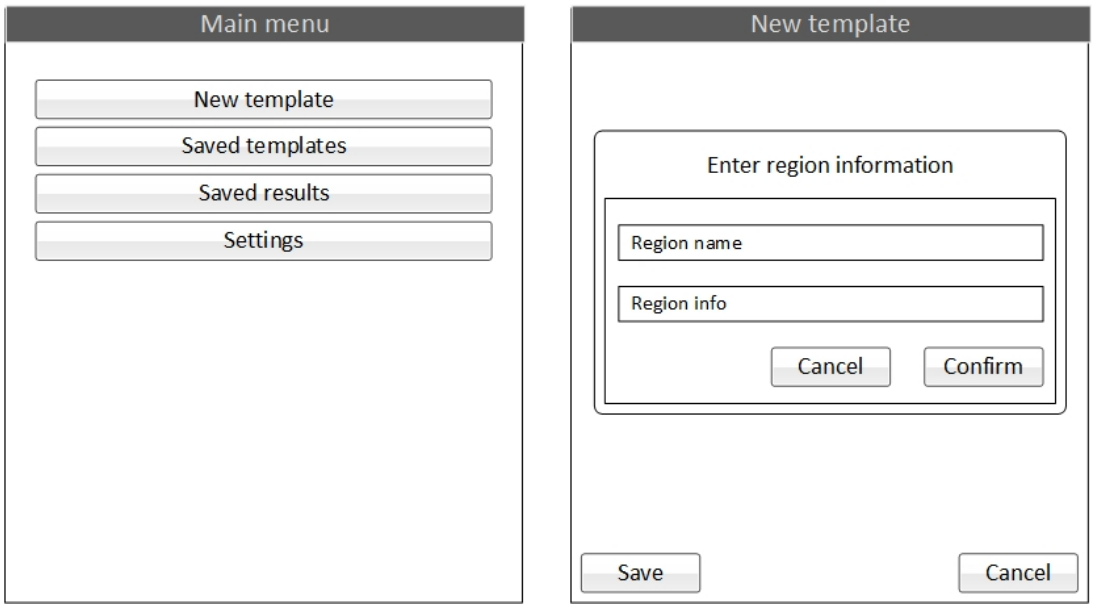
4.6 Triedny diagram



Obr. 8: Triedny diagram aplikácie

Triedny diagram znázorňuje štruktúru a vzťahy medzi aktivitami a triedami aplikácie.

4.7 Užívateľské rozhranie



Obr. 9: Wireframe vybraných aktivít

Ukážka užívateľského rozhrania aktivít hlavného menu a vytvorenia čítaného regiónu.

5 Implementácia

Nasledujúca kapitola popisuje implementáciu Android aplikácie.

5.1 Hlavná aktivita

Po spustení aplikácie sa načíta hlavná aktivita, v ktorej prebieha inicializácia externých knižníc, načítanie nastavení a konfigurácia aplikácie. Nastavenia aplikácie sú uložené pomocou triedy SharedPreferences a obsahuje dáta ako napríklad úroveň transparentnosti náhľadu pri používaní šablóny alebo úroveň prahovania.

Inicializácia knižnice tessTwo prebieha v dvoch fázach. Prvá fáza zahŕňa kopírovanie potrebných súborov do vopred definovaného priečinku v pamäti zariadenia. Tieto súbory sú nevyhnutné pre správne fungovanie rozpoznávania textu, pretože obsahujú naučené dáta ako prototypy znakov a podobne. Druhá fáza inicializácie spočíva vo vytvorení API objektu a volaní funkcie *init()*. Parametrami sú absolútna cesta k súboru s prototypmi a jazyk používaného súboru. Následne môžeme objekt používať na rozpoznávanie textu, je však dobré nastaviť správny mód PSM, ktorý teoreticky zvyšuje pravdepodobnosť správneho rozpoznania. Knižnica openCV potrebuje pre správne fungovanie v niektorých prípadoch externú aplikáciu, ktorá zaisťuje správne fungovanie naprieč Android zariadeniami. V prípade, že v systéme chýba, užívateľ je vyzvaný na jej stiahnutie z Obchodu Play. Následne sa knižnica inicializuje asynchrónne, o jej úspešnosti nás informuje výsledok získaný v callbacku.

Po úspešnej inicializácii môžeme s aplikáciou pracovať, zobrazí sa nám hlavné menu, z ktorého môžeme pokračovať do nastavení, prezerať si existujúce šablóny a výsledky rozpoznávania alebo vytvoriť novú šablónu.

5.2 Aktivita fotoaparátu

Základná aktivita fotoaparátu je výrazne limitovaná svojou funkcionalitou a je preto potrebné implementovať vlastnú aktivitu na zachytávanie obrázkov. Okrem samotného obrázka potrebujeme pridať polotransparentnú vrstvu, v ktorej bude umiestnená predloha obrázka z používanej šablóny. Pri obyčajnom fotení bude táto vrstva plne priehľadná, pri používaní šablóny sa zobrazí – úroveň transparentnosti môžeme meniť v nastaveniach.

Jednotlivé vrstvy užívateľského rozhrania aktivity fotoaparátu znázornené obrázkom 10 sú na sebe naskladané tak, že najspodnejšia vrstva vykresľuje aktuálny obraz zachytávaný fotoaparátom pomocou SurfaceHolder. Nasleduje transparentná vrstva šablóny zobrazujúca obrázok pomocou ImageView a najvrchnejšia vrstva obsahuje UI elementy ako tlačidlá.

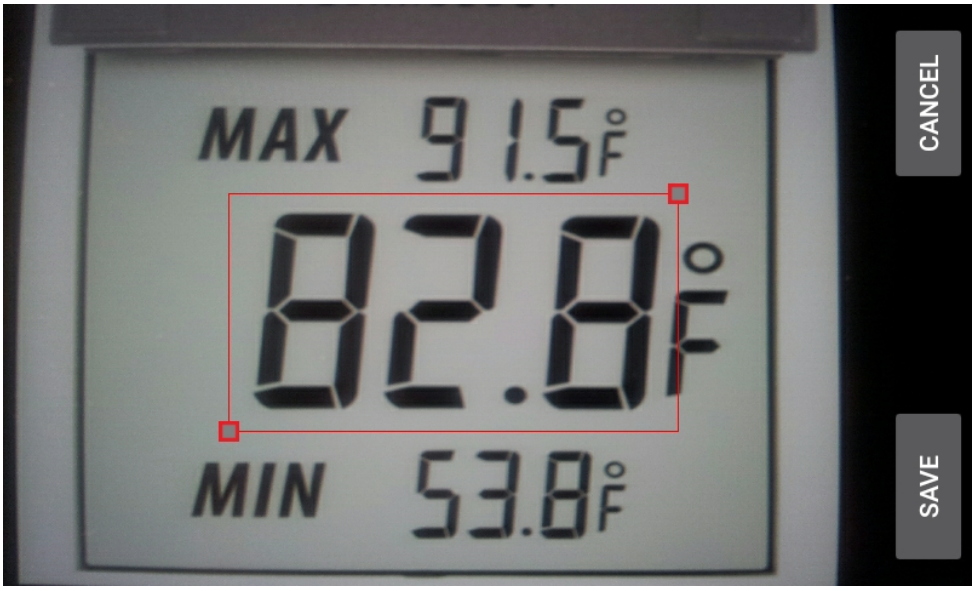
Staršie a menej výkonné Android zariadenia môžu mať problém pri práci s obrázkami, kedy je limitom pamäť RAM. Preto musí vedieť fotoaparát pracovať v móde kompatibility, kedy sa rozlíšenie obrázku limituje na rozlíšenie displeja. Novšie a výkonné zariadenia používajú najvyššie možné rozlíšenie fotoaparátu.

Obr. 10: Vrstvy užívateľského rozhrania aktivity fotoaparátu

Odfotený obrázok sa nám vráti ako bajtové pole, ktoré dekódujeme na Bitmap a následne uložíme vo formáte JPEG do pamäte zariadenia. Výsledok aktivity je absolútna cesta k obrázku.

5.3 Vytvorenie šablóny a vyznačenie regiónov

Odfotený obrázok môžeme použiť na vytvorenie šablóny. Šablóna pozostáva z obrázkovej predlohy a regiónov, z ktorých sa budú čítať relevantné dáta. Aktivita vytvorenia šablóny pozostáva z obrázkovej predlohy a selektora regiónu znázorneného obrázkom 11, ktorý je umiestnený vo vrstve nad predlohou. Na vykreslenie vyznačeného regiónu sa používa Canvas, ktorý najskôr vyplníme transparentnou farbou a následne vykreslíme body a čiary selektoru. Pre posun bodu selektoru je najskôr potrebné vypočítať, ktorý bod sa má presunúť. Bod posunu určíme na základe vzdialeností bodu, ktorý je zaznamenaný pri dotyku displeja a jednotlivých bodov selektoru. Na výpočet vzdialenosti bodov sa používa Euklidova metrika. Po presunutí bodov na požadované miesto pokračujeme zadaním názvu a popisu regiónu do vstupného dialógu a proces opakujeme, kým nie sú vyznačené všetky regióny, z ktorých chceme čítať. Výslednú šablónu môžeme uložiť serializovaním do dátového súboru alebo zahodiť.



Obr. 11: Ukážka vyznačenia regiónu

5.4 Používanie šablóny

Pred samotným použitím šablóny na rozpoznávanie textu musíme vybrať požadovanú šablónu zo zoznamu všetkých šablón. Tie sú zobrazené pomocou ListView a príslušného ArrayAdapter. Zoznam šablón obsahuje okrem názvu aj počet regiónov. Po vybratí regiónu sa zobrazí aktivita náhľadu šablóny, v ktorej sú graficky vyznačené jednotlivé regióny.

Obr. 12: Ukážka existujúcej šablóny

V prípade, že sa rozhodneme šablónu použiť, sa spustí aktivita fotoaparátu s parametrami pre zobrazenie transparentej vrstvy. Po odfotení je potrebné obrázok rozdeliť na regióny, ktoré budú poslané na predspracovanie a následné rozpoznávanie. Veľkosť jednotlivých regiónov sa počíta na základe percentuálnej veľkosti viazanej k šablóne, čo zaistí správne rozdelenie obrázka nezávisle na jeho veľkosti a pomere strán. Pre každý región je vytvorený objekt Bitmap, ktorý sa pred spracovaním uloží do pamäte zariadenia, nakoľko nie je možné presúvať veľké množstvo dát medzi aktivitami v podobe poľa bajtov. Po ich uložení nasleduje predspracovanie.

5.5 Predspracovanie obrázka

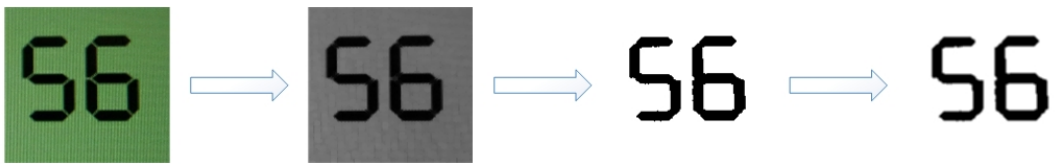
Pre zvýšenie úspešného rozpoznávania potrebujeme obrázok predspracovať, čo zaistí odstránenie vizuálnych artefaktov. Hlavný problém pri rozpoznávaní textu 7 segmentových displejov je oddelenie jednotlivých segmentov, ktoré tvoria každý znak. Metódou pokus-omyl bola nájdená optimálna kombinácia operácií knižnice openCV, ktorá konzistentne produkuje výsledné obrázky v dobrej kvalite a bez vizuálnych artefaktov.

Najskôr musíme odfotený obrázok previesť do čierno-bielej farebnej škály, pretože Tesseract pracuje interne práve s takýmito obrázkami. Taktiež máme kontrolu nad tým, ako obrázok spracujeme a nenechávame túto povinnosť externej knižnici. Túto konverziu zaistí funkcia cvtColor s parametrom COLOR_RGB2GRAY.

Následne aplikujeme morfologickú transformáciu morphologyEx, ktorej parameter MORPH_OPEN spôsobí vyplnenie prázdnych miest medzi segmentmi znaku.

Takýto obrázok je možné previesť do binárnej formy, čo znamená použitie 2 farieb, čiernej a bielej. Táto operácia sa nazýva prahovanie a docielime ju funkciou threshold, ktorej parametre určujú, ktorá časť čierno-bielej škály sa má konvertovať na bielu a ktorá na čiernu.

Poslednou operáciou predspracovania je rozmazanie obrázka, ktoré odstráni malé defekty ako výbežky alebo dierky a celkovo zjemní výsledný obrázok, ktorý vyzerá prirodzenejšie. Obrázok je teraz pripravený na rozpoznávanie textu. [13]



Obr. 13: Ukážka procesu predspracovania

5.6 Rozpoznávanie textu

Pripravené obrázky môžeme poslať na rozpoznávanie. To pozostáva z vytvorenia API objektu, ktoré máme nainicializované od spustenia aplikáce. Najskôr objektu nastavíme Bitmap regiónu, ktorý chceme rozpoznávať. Následne zavolaním funkcie getUTF8Text získame výsledok rozpoz-

návania vo forme znakového reťazca. Z neho ešte odstránime prázdne znaky a máme k dispozícii výsledné dáta, ktoré môžeme ďalej uložiť alebo odoslať.

```
TessBaseAPI api = OcrHelper.Instance().GetOcrApi();
for (int regionIndex = 0; regionIndex < regionCount; regionIndex++)
{
    Bitmap bmpToOcr = GetRegionBitmap(regionIndex);
    if (bmpToOcr != null)
    {
        api.setImage(bmpToOcr);
        capture.RegionInfos.get(regionIndex).Value = api.getUTF8Text();
    }
}
api.end();
```

Výpis 1: Ukážka kódu rozpoznávania textu

5.7 Uloženie a zdieľanie dát

Výstupné dáta z rozpoznávania sa automaticky ukladajú do pamäte zariadenia. Pri prehliadaní výsledkov môžeme jednotlivé výsledky zaslať na URL definovanú v šablóne. Uložené výsledky sú vo forme binárneho súboru z ktorého sa pri odosielaní generujú JSON dáta.

6 Testovanie a ladenie

Po naimplementovaní ostáva aplikáciu dôsledne otestovať a doladiť problémové časti. Práve táto časť zabrala veľké množstvo času, pretože sme pri testovaní narazili na niekoľko problémov.

6.1 Prvé výsledky

V základnej konfigurácii používa knižnica Tesseract preddefinovaný súbor prototypov určený na rozpoznávanie znakov používaných v bežných anglických textoch. Podporuje tiež whitelist jednotlivých znakov, ktorý je aplikovaný pri rozpoznávaní. Na testovanie bol použitý vstupný obrázok 14.

Obr. 14: Ukážka vstupu prvého testu

Úspešnosť testu bola 0% bez ohľadu na to, či bol whitelist použitý alebo nie. Bez whitelistu bol výsledný reťazec naplnený zdanlivo náhodnými znakmi. S ním sa vo výsledku objavili čísla, ktoré ani zďaleka nepripomínali očakávaný výsledok.

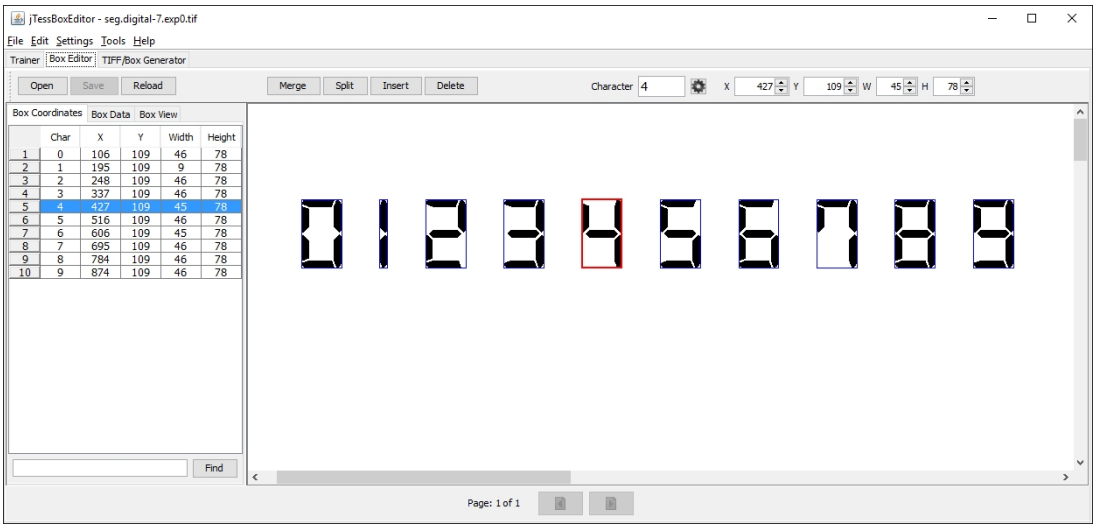
6.2 Trénovanie OCR a použité nástroje

Po neúspechu prvotného testovania bolo jasné, že základný súbor prototypov je nedostačujúci pre potreby projektu. Bolo potrebné natrénovať nový typ písma, ktorý je typický pre sedem segmentové displeje. Trénovanie je do určitej miery automatizovateľné pomocou voľne dostupných nástrojov, zvyšná časť musí byť vykonaná manuálne.

Trénovanie začína vyhľadaním typu písma, ktorý je najpodobnejší s čítanou predlohou. Na internete nájdeme niekoľko typov písma, ktoré je možné použiť na trénovanie, v našom prípade to bol Digital-7 [14]. Po stiahnutí a nainštalovaní fontu vytvoríme vstupný textový súbor, ktorého obsah sú znaky určené na učenie OCR. Následne pomocou nástroja jTessBoxEditor [15] vygenerujeme obrázok vo formáte TIFF obsahujúci vstupný text vykreslený požadovaným fontom. Tento obrázok je dôležitý trénovací prvok a závisí od neho úspešnosť celého trénovania. Musíme dbať na to, aby sa jednotlivé znaky v obrázku neprekrývali, čo by spôsobilo chyby neskôr pri trénovaní.

Z vygenerovaného obrázku potrebujeme vytvoriť BOX súbor, ktorý obsahuje dáta popisujúce umiestnenie znakov v obrázku a ich hodnotu. Tento súbor vytvárame pomocu konzolového rozhrania knižnice Tesseract, kde špecifikujeme názov vstupného súboru, trojmiestny kód jazyka v ktorom bude rozpoznávaný text a parametre *batch.nochop makebox*. Tu nastáva prvý závažný

problém, pretože knižnica Tesseract nie je veľmi výrečná a väčšina chýb je oznámená správou *Page Empty!!*. Z tejto správy sa dá len ťažko určiť, kde nastal problém. Častým problémom býva malá vzdialenosť medzi znakmi, inokedy je znak rozdelený na viacero častí, ktoré sú príliš blízko pri sebe. Po odstránení chýb pokračujeme kontrolou vygenerovaného BOX súboru pomocou programu jTessBoxEditor, kde skontrolujeme správnosť umiestnenia znakov a ich hodnôt. Kedže stále používame základný súbor prototypov, je pravdepodobné, že hodnoty znakov sú priradené nesprávne a musíme ich opraviť.



Obr. 15: Nástroj na úpravu BOX súborov jTessEditor

Existujúci pár súborov *obrázok - BOX* následne použijeme pri trénovaní, ktoré spustíme cez konzolové rozhranie Tesseract s parametrom box.train. Výsledkom tejto operácie je TR súbor, ktorý obsahuje rysy znakov. V prípade, že sme nesprávne vygenerovali BOX súbor, sa vo výstupe zobrazia chybové správy APPLY_BOX, tentokrát s detailnejším popisom. Z BOX súboru extrahujeme množinu všetkých rozpoznateľných znakov pomocou nástroja unicharset_extractor. Jeho výsledkom je súbor unicharset. Súbory unicharset a TR použijeme v nástrojoch mftraining a cntraining, ktoré vygenerujú zvyšok potrebných súborov.

V tomto momente máme vygenerované všetky potrebné súbory. Ostáva ich spojiť do výsledného súboru prototypov. Ten vytvoríme umiestnením súborov do rovnakého priečinku a spustíme nástroj combine_tessdata, ktorého parameter je trojmiestny kód jazyka. V prípade, že nenastali žiadne chyby, bude vygenerovaný prototypový súbor TRAINEDDATA.

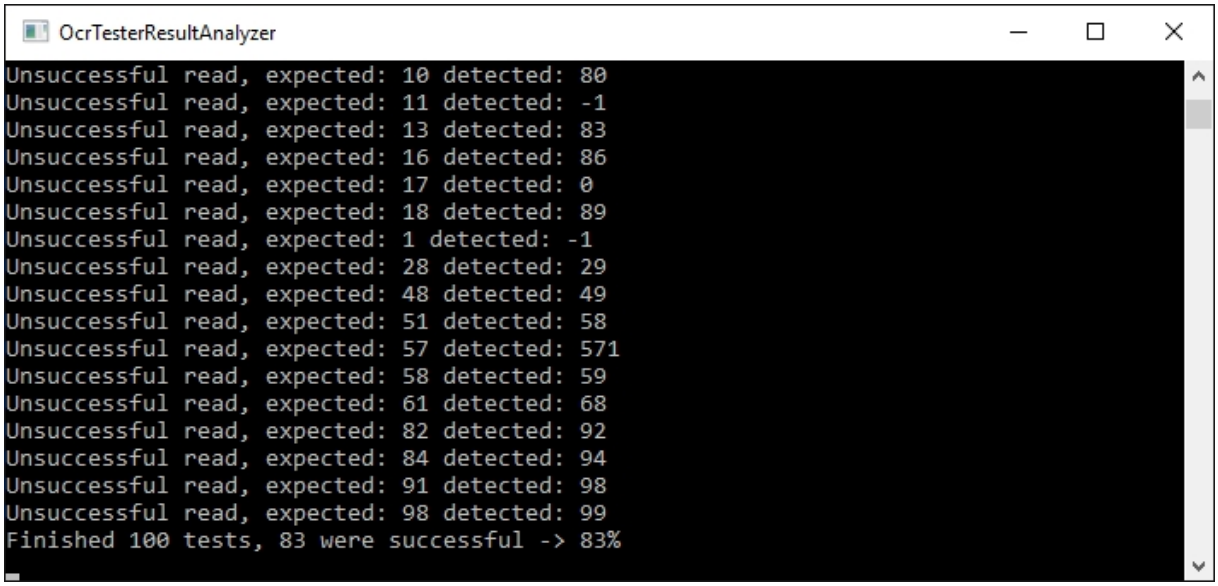
6.3 Automatizované testovanie

Pre dôkladné otestovanie rozpoznávania je potrebné vykonať stovky až tisíce testov. Manuálne testovanie je nepraktické a zdĺhavé, je preto potrebné vytvoriť mechanizmus, ktorý bude vykonávať testy automaticky a po vykonaní testu vyhodnotí jeho výsledky. Testovací mechanizmus bude pozostávať z dvoch komponentov, Android aplikácie a simulátora 7-segmentového displeja,

ktoré budú navzájom prepojené cez Socket komunikujúci na lokálnej sieti. Simulátor je desktopová aplikácia generujúca náhodné čísla, ktoré sú následne odfotené a rozpoznané cez Android aplikáciu. Okrem generovania čísiel potrebujeme meniť farby čísiel a ich pozadia, aby sme otestovali rôzne typy displejov. Výsledok rozpoznávania sa odosiela do simulátora, ktorý porovná vygenerované číslo s prečítanou hodnotou a výsledok uloží do pamäte. Po určenom množstve vykonaných testov sa testovanie zastaví a zobrazia sa výsledky úspešnosti spolu s nesprávne identifikovanými hodnotami, na ktoré treba klásť dôraz pri ďalšom trénovaní OCR.

Obr. 16: Nástroj generujúci náhodné čísla pre OCR

6.4 Lepšie výsledky

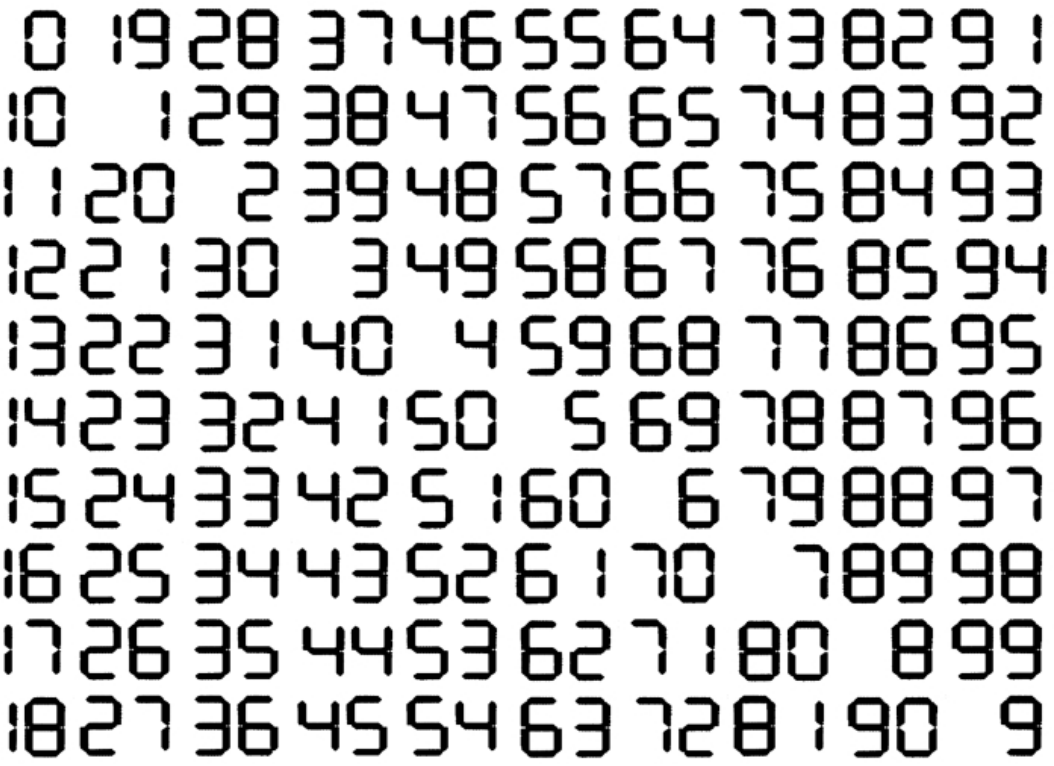


Obr. 17: Nástroj na vyhodnotenie výsledkov testu

S natrénovaným fontom môžeme pokračovať v testovaní. Súbor prototypov umiestnime do zložky tessdata a doplníme rozpoznávanie konzolového rozhrania Tesseract o parameter -l seg, kde seg je náš trojmiestny kód jazyka. Nasledujúci test odhalil určité nedostatky hlavne v rozlišovaní párov znakov 1-8 a 8-9. Úspešnosť sa výrazne zvýšila, z 0 na 83%. Trénovanie OCR prinieslo očakávané výsledky, stále je ale miesto na zlepšenie a budeme v trénovaní pokračovať.

6.5 Lepšie trénovanie OCR

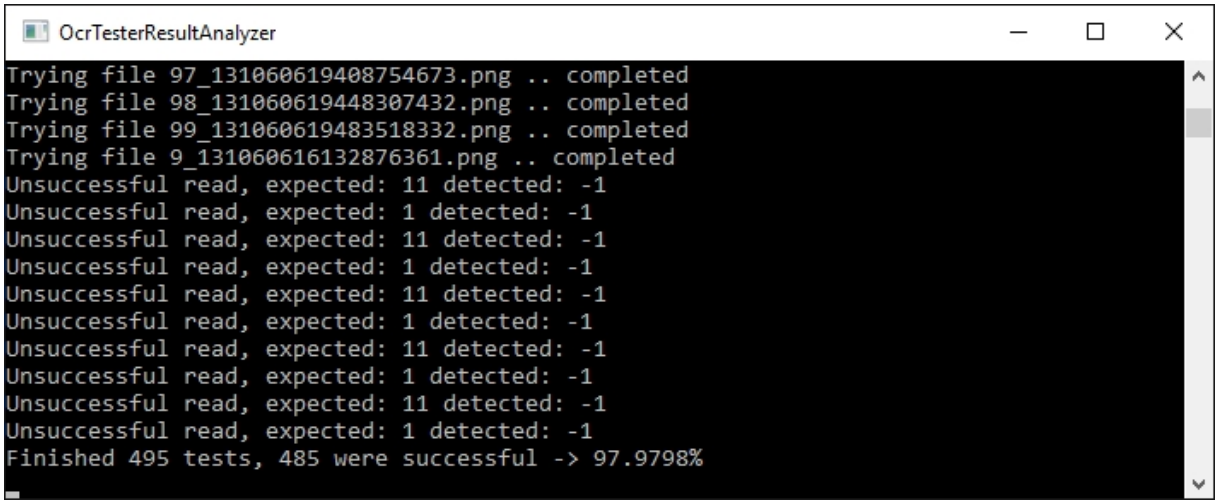
Pomocou automatizovaného testovania sme získali informácie, ktoré znaky sú problémové. Taktiež máme k dispozícii množstvo zachytených a predspacovaných obrázkov, ktoré boli použité pri rozpoznávaní. Tieto obrázky sa mierne líšia od ich vygenerovanej predlohy a obsahujú rôzne variácie, ktoré môžu v reálnom rozpoznávaní nastať. Tieto predspracované obrázky môžeme použiť na trénovanie OCR za účelom rozšírenia množiny rysov znakov, čo pozitívne ovplyvní úspešnosť rozpoznávania. Predspracované obrázky spojíme do jedného veľkého súboru, s ktorým následne OCR trénujeme.



Obr. 18: Predspracované obrázky určené na trénovanie

6.6 Konečné výsledky

V konečnom teste bolo spracovaných 495 obrázkových vstupov s rôznymi úrovňami kontrastu, farby textu a pozadia. Výsledná úspešnosť dosiahla prijateľných 97%, problémový znak stále ostáva ‘1’.



Obr. 19: Výsledky testovania rôznych displejov

Pravdepodobnou príčinou nesprávneho rozpoznávania tohto znaku je jeho malá šírka v porovnaní s ostatnými znakmi. Tento problém nie je možné jednoznačne odstrániť, pretože Tesseract pri rozpoznávaní nevypisuje žiadne chyby. Ostatné znaky sú rozpoznané správne vo všetkých prípadoch. Testy boli vykonané pri základnom nastavení prahovania, prípadné problémy predspracovania odstránime zmenou úrovne prahovania.

Záver

V bakalárskej práci sme navrhli a implementovali aplikáciu na rozpoznávanie dát z LCD displejov. Táto aplikácia má minimálne požiadavky na spustenie a môžeme ju využívať na väčšine Android zariadení s fotoaparátom. Pri implementácii sme využili voľne dostupné knižnice openCV na predspracovanie obrázka, Tesseract a tessTwo na rozpoznávanie znakov.

Aplikácia používa na rozlíšovanie rôznych displejov systém šablón. Šablóna obsahuje obrázkovú predlohu, ktorú používame na zarovnanie fotoaparátu a displeja. Každá šablóna má svoj unikátny názov a pole obdĺžnikových regiónov, z ktorých pomocou OCR čítame dáta. V šablóne môžeme špecifikovať URL, na ktorú dáta v prípade potreby odošleme.

Odfotené obrázky bolo pred rozpoznávaním potrebné predspracovať. Na predspracovanie sme využili knižnicu OpenCV. Pomocou nej sme na vstupný obrázok aplikovali sériu operácií, ktoré z obrázka odstránili vizuálne artefakty. Výsledkom predspracovania bol dvojfarebný obrázok pripravený na rozpoznávanie.

Pri rozpoznávaní 7-segmentových displejov základný súbor prototypov nestačil. Pomocou pribalených a voľne dostupných nástrojov sme knižnicu Tesseract naučili rozpoznávať novú znakovú sadu Digital-7. Pomocou testovania sme identifikovali problémové znaky a vytvorili nový prototyp znakov, ktorý lepšie zodpovedal čítanému vstupu. Tento prototyp výrazne zlepšil úroveň rozpoznávania a umožnil automatizáciu ďalšieho trénovania a testovania.

Úspešnosť rozpoznávania je dostatočne vysoká pre praktické využitie aplikácie. Tú je možné jednoduchým spôsobom rozšíriť o rozpoznávanie ďalších znakov a znakových sád zámenou súboru prototypov. Praktické využitie aplikácie zahŕňa automatizované čítanie a archiváciu dát zo zariadení ako tlakomery, meteostanice a podobne.

Aplikácia v čase písania práce nie je dostupná na stiahnutie, v budúcnosti sa plánuje jej vydanie cez platformu Google Play.

Literatura

[1] ISO 1073-1:1976: Alphanumeric character sets for optical recognition – Part 1: Character set OCR-A – Shapes and dimensions of the printed image. ISO - International Organization for Standardization [online]. [cit. 2016-04-27]. Dostupné z: http://www.iso.org/iso/catalogue_detail.htm?csnumber=5567

[2] ISO 1073-2:1976: Alphanumeric character sets for optical recognition – Part 2: Character set OCR-B – Shapes and dimensions of the printed image. ISO - International Organization for Standardization [online]. [cit. 2016-04-27]. Dostupné z: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=5568

[3] OCR-A vs OCR-B. Quora [online]. [cit. 2016-04-27]. Dostupné z: https://qph.is.quoracdn.net/main-qimg-c1648f10e3377122db73255b9ecf8cd6?convert_to_webp=true

[4] Optical Character Recognition [online]. [cit. 2016-04-28]. Dostupné z: https://www.nr.no/ eikvil/OCR.pdf

[5] WOODFORD, Chris. Optical character recognition (OCR) [online]. 2016 [cit. 2016-04-26]. Dostupné z: http://www.explainthatstuff.com/how-ocr-works.html

[6] Tesseract Open Source OCR Engine [online]. [cit. 2016-04-25]. Dostupné z: https://github.com/tesseract-ocr/tesseract‘

[7] Python-based tools for document analysis and OCR [online]. [cit. 2016-04-25]. Dostupné z: https://github.com/tmbdev/ocropy

[8] Asprise OCR and Barcode Recognition [online]. Asprise Software, 2016 [cit. 2016-04-25]. Dostupné z: https://asprise.com/royalty-free-library/ocr-api-for-java-csharp-vb.net.html

[9] Fork of Tesseract Tools for Android. [online]. [cit. 2016-04-25]. Dostupné z: https://github.com/rmtheis/tess-two

[10] ANNUZZI, Joseph. Introduction to Android Application Development. 5. Addison-Wesley, 2016. ISBN 978-0134389455.

[11] Android Distribution Updated for April 2016: Marshmallow Closes in on 5%. Droid Life [online]. 2016 [cit. 2016-04-27]. Dostupné z: http://www.droid-life.com/2016/04/05/android-distribution-update-for-april-2016/

[12] SMYTH, Neil. Android Studio Development Essentials - Android 6 Edition. CreateSpace, 2015. ISBN 978-1519722089.

[13] MUHAMMAD, Amgad. OpenCV Android Programming By Example. Packt Publishing, 2015. ISBN 978-1783550593.

[14] Digital 7 Font [online]. [cit. 2016-04-28]. Dostupné z: http://www.dafont.com/digital-7.font

[15] JTessBoxEditor [online]. [cit. 2016-04-26]. Dostupné z: http://vietocr.sourceforge.net/training.html

Zoznam príloh

Na priloženom CD sa nachádzajú nasledujúce prílohy:

Tabuľka 2: Popis štruktúry CD

Priečinok	Popis
AndroidApplication	zdrojový kód Android aplikácie
TestingApplication	zdrojový kód testovacej aplikácie
Images	vstupné a predspracované obrázky
Apk	kompilovaná Android aplikácia